TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

PROGRAMACION II

NOVIEMBRE 2025

TRABAJO PRACTICO INTEGRADOR

Sistema de gestión de Mascotas y Microchips

INTEGRANTES Y ROLES GRUPO N°102:

Barroeta Victor – Comisión: 2 – DNI: 95805903: Tester QA y Entities

Barrutia Milagros – Comisión:10 – DNI: 38603968: Base de datos y App Menú

Batista Federico – Comisión:10 – DNI: 38019234: Services y Congiguracion

Benitez Rodrigo – Comisión:10 – DNI: 38778097: Dao y Entities

En el presente informe sobre el Proyecto Integrador de Programación II se presentará el desarrollo de la creación de un sistema de gestión de Mascotas, vinculadas a un microchip. Se creará una base de datos ficticia para su prueba de funcionamiento y se aplicará para su creación lo estudiado en la materia: Programación Orientada a Objetos y Conexión con Bases de Datos. Para aplicar estos conocimientos se utilizará el Lenguaje Java para POO y SQL para la base de datos.

ELECCION DE DOMINIO

En nuestro proyecto integrador nuestro dominio es la gestión de registro de mascotas, utilizando un microchip. El propósito es que una institución lleve un control de la población de mascotas en una zona, y en el caso que una mascota se pierda se podrá acceder rápidamente a los datos del dueño.

Con el registro se podrá acceder a datos como localidad, dueño, raza, observaciones y la veterinaria que registro al animal.

DISEÑO

La relación entre mascota y microchip es del tipo 1 a 1, en este caso, **una** instancia del objeto está relacionada con una instancia del otro objeto. Esto se define como una relación unidireccional entre mascota y microchip.

Este tipo de relaciones se implementan estableciéndole a la clase Mascota un atributo llamado microchip que establecerá la asociación unidireccional entre ambos, con su respectivo Setter.[1]

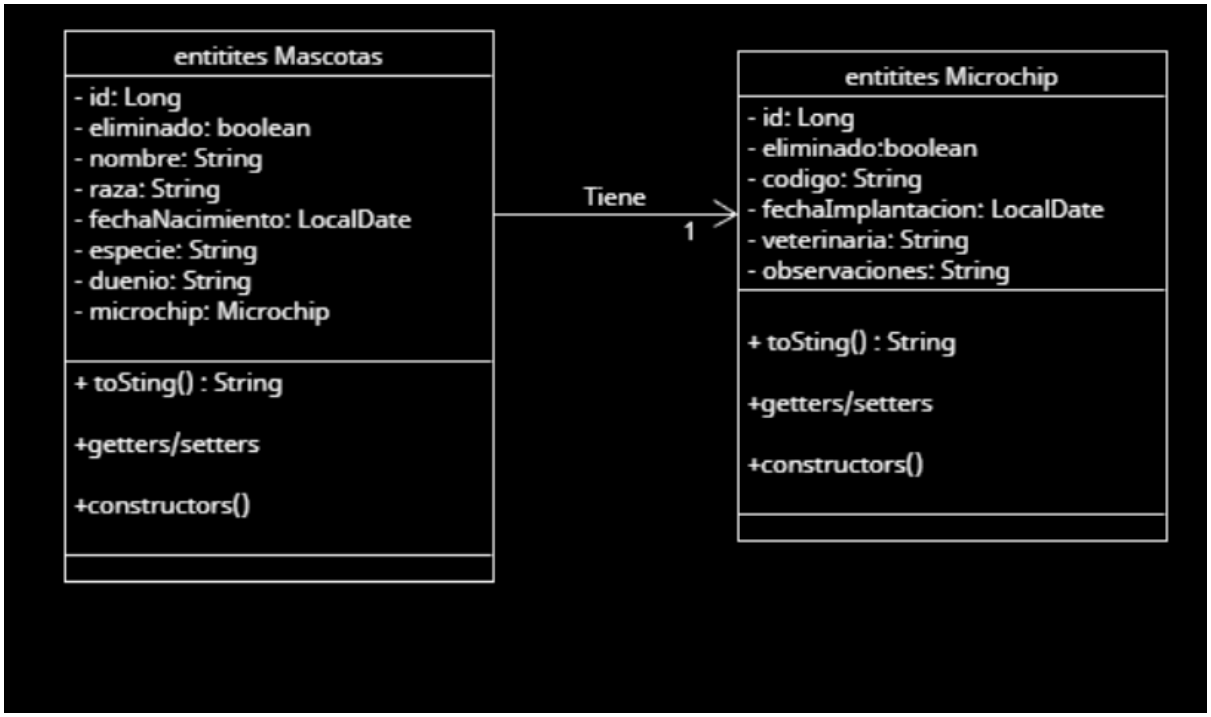
Para trabajar la relación entre las 2 entidades elegimos la Clave Foránea Única en la tabla B (Microchip), entonces la tabla microchip contiene una columna mascota_id que actúa como Clave Foránea apuntando al id de la tabla mascota.

Para asegurar que un microchip no se asigne a más de una mascota (y viceversa), añadimos una restricción UNIQUE a la columna mascota_id en la tabla microchip.

Elegimos este método porque permite que la Mascota y el Microchip tengan sus propios ciclos de vida e IDs independientes, lo cual es más flexible.

A continuación, se presenta el UML con las clases mencionadas y sus respectivos atributos.

[1] Material de la catedra



PERSISTENCIA

La persistencia de los datos en el sistema se gestionó mediante la API JDBC (Java DataBase Connectivity), permitiendo una comunicación directa y eficiente con la base de datos . Para garantizar la integridad y consistencia de los datos, se implementó un control de transacciones de manera explícita.

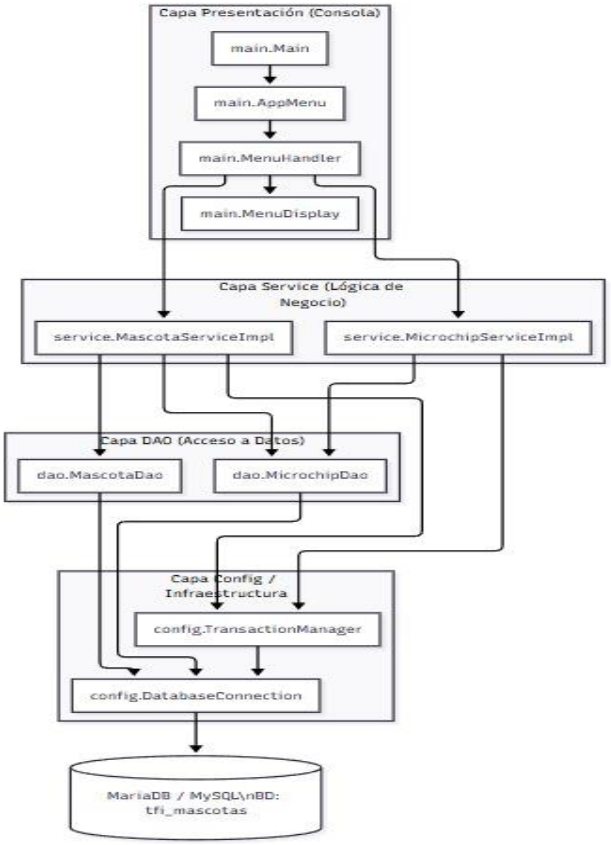
En un entorno JDBC estándar, cada instrucción SQL realiza un *commit* automático, lo que resulta problemático cuando una unidad de trabajo requiere múltiples operaciones. Para asegurar el principio de Atomicidad, se diseñó la clase TransactionManager.

Este proceso se explica en detalle en la capa Configuración.

LOGICA DE NEGOCIO

- Cada mascota debe tener un id del tipo numérico único e irrepetible.
- Cada Microchip debe tener un id del tipo numérico único e irrepetible.
- Una mascota solo puede tener asociado un único microchip, que no debe coincidir con el de otra mascota.
- Una mascota no puede tener más de un dueño.
- La fecha de implementación del microchip no puede ser posterior a la fecha actual.
- Todos los campos son obligatorios.

ARQUITECTURA DE CAPAS



El diseño del proyecto se armó de manera secuencial. A continuación, se explicarán las capas que se fueron programando a medida que transcurría el mismo. Se adjunta un esquema con la relación entre las capas

En la imagen se muestra como desde el Menú, que se podría decir que es la parte visible para el usuario cuando se opera se va conectando con las distintas capas hasta llegar a la Base de Datos (tfi_mascotas). Cuando el usuario realiza una operación desde el menú, se pasa por la Capa Service para validar la lógica de negocio, explicada posteriormente. Para ejecutar esa capa se necesita el código de la capa DAO, la cual va a interactuar con la base de datos. Y por último la capa Config va a realizar la conexión con la base de datos.

ENTITIES

En esta capa se crean las clases Mascota, Microchip y la capa base, una clase madre que tendrá atributos que heredaran Mascota y Microchip (id y eliminado). Luego se crean las clases hijas, Mascota y Microchip, con sus respectivos atributos y métodos Getter y Setters. Además del constructor. Esta capa va a estar relacionada con todas las capas posteriores ya que utilizaran sus métodos.

Vale aclarar que en este momento se crea la base de datos en SQL, se crean 2 tablas (mascota y microchip) con sus atributos y se insertan registros ficticios para luego probar el funcionamiento.

CONFIGURACION

La capa de configuración es la encargada de administrar todos los aspectos relacionados con la conexión a la base de datos. Se crea una clase DataBase Connection con métodos estáticos, uno de esos métodos sera el creador de la conexión con la base de datos:

```
public static Connection getConnection() throws SQLException {
    return DriverManager.getConnection(URL, USER, PASSWORD);
}
```

Aquí se define un método que recibe como parámetro una URL, un usuario y una contraseña, y que será invocado cada vez que alguna capa del proyecto necesite establecer una conexión con la base de datos.

Dentro de esta misma capa también se implementa el control de transacciones mediante la clase **TransactionManager**.
En JDBC, cada operación realiza un *commit* automático; sin embargo, cuando varias operaciones deben ejecutarse de forma conjunta (por ejemplo, insertar una mascota y su microchip), es necesario agruparlas dentro de una transacción para garantizar la integridad de los datos.

Por este motivo, la clase TransactionManager permite controlar manualmente el *commit*, ejecutar un *rollback* si ocurre algún error y asegurar el cierre adecuado de la conexión.

Todo este funcionamiento es posible gracias a **JDBC (Java DataBase Connectivity)**, la API que conecta Java y SQL, permitiendo ejecutar consultas, manejar transacciones y comunicarse con la base de datos de manera estandarizada.

DAO

Es la intermedia entre la base de datos con el sistema. y se encarga de la persistencia de los registros de la base de datos. Utilizamos el conjunto estandarizado de operaciones: los métodos CRUD (crear, leer, actualizar y eliminar). Gracias a esta capa se podrá interactuar con la base de datos y realizar tareas esenciales de manipulación de datos. Para lograr esto se crean 3 clases: una general y una para cada clase especifica, en nuestro caso Mascota y Microchip. En la clase general se definen los métodos CRUD mencionados anteriormente y en cada clase especifica se implementan esos métodos. Primero se definen los comandos SQL para las

operaciones específicas de cada entidad. Y luego se invocan a los métodos CRUD aplicando esos comandos SQL.

SERVICIOS (Orden de transacciones)

La capa Services utiliza los métodos definidos en la capa DAO para realizar las operaciones, pero agregando validaciones y reglas de negocio específicas de cada entidad. Estas reglas pueden incluir controles como: verificar que un campo no esté vacío, que no existan duplicados, que los datos cumplan con un formato determinado, etc.

Esta lógica es particular de cada sistema y es lo que comúnmente llamamos lógica de negocio.

En esta capa se define el orden de las transacciones:

1. **TransactionManager txManager = new TransactionManager()**: Se obtiene una nueva conexión a la BD.
2. **txManager.startTransaction()**: Se llama a conn.setAutoCommit(false).
3. **try { ... }**: Inicia el bloque transaccional.

- **Paso A:** Se llama a mascotaDao.crear(mascota, conn). El DAO (A) inserta la mascota y setGeneratedId actualiza el ID en el objeto mascota.
- **Paso B:** Se llama a microchipDao.crear(chip, mascota.getId(), conn). El DAO (B) usa el ID de la mascota (Paso A) para insertar el microchip, cumpliendo la FK.

4. **txManager.commit()**: Si A y B tuvieron éxito, se confirman los cambios en la BD.
5. **catch (Exception e) { ... txManager.rollback(); }**: Si cualquier paso falla (ej. el código del chip estaba duplicado), se captura la excepción y se llama a rollback(), revirtiendo la creación de la Mascota (Paso A).

finally { ... txManager.close(); }: Pase lo que pase, la conexión se cierra de forma segura.

De esta manera, la capa Services coordina la operación, aplica reglas y delega el acceso a la base de datos a la capa DAO.

MENU (Validaciones de entrada)

En esta última capa se implementan todas las capas anteriores. Se creará el menú que el usuario utilizará para poder interactuar con el programa. Se utilizará la clase estudiada Scanner para registrar las entradas del usuario y se conectaran con la base de datos para poder hacer eficiente su solicitud.

Se crea una clase menú que utiliza un ciclo Do While, y se valida la entrada con un número y se cierra el ciclo utilizando el 0. Dentro del ciclo While se utilizará el comando switch y se definen los casos para cada opción.

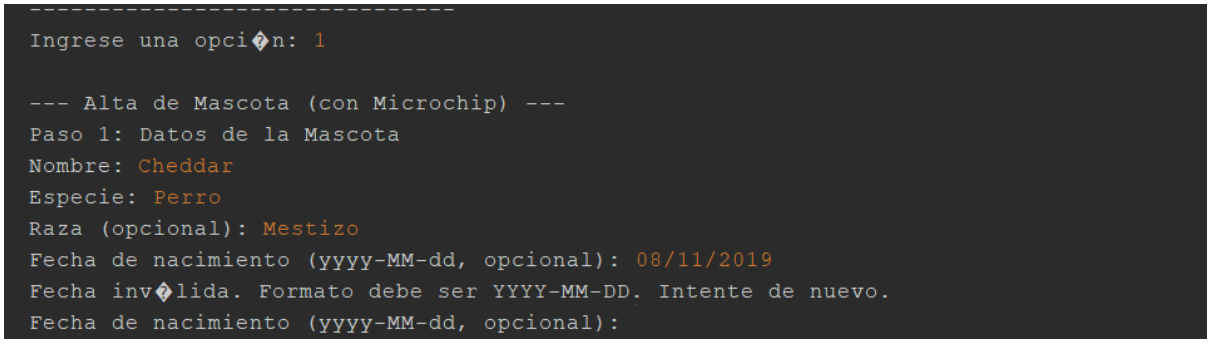
Para validar las entradas se usan bucles try-catch para NumberFormatException y DateTimeParseException para asegurar que el usuario ingrese números (leerLong) y fechas (leerFechaOpcional) válidas.

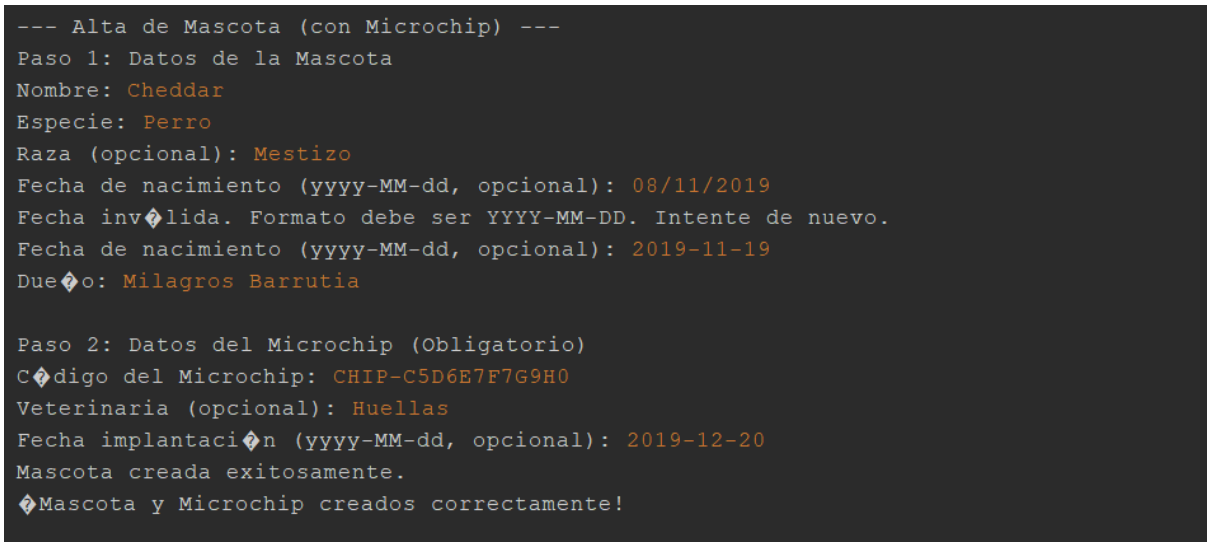
Ademas, se implementó una "salida de emergencia": el usuario puede escribir CANCELAR en cualquier campo para abortar el proceso (CancelActionException).

PRUEBAS REALIZADAS

Se adjuntan capturas de pantalla de distintas operaciones y sus validaciones

1. Se da de alta una mascota





2. Prueba de unicidad de Microchip

```
--- Alta de Mascota (con Microchip) ---
Paso 1: Datos de la Mascota
Nombre: Ricota
Especie: Gato
Raza (opcional): Mestizo
Fecha de nacimiento (yyyy-MM-dd, opcional): 2022-10-15
Due□o: Milagros Barrutia

Paso 2: Datos del Microchip (Obligatorio)
C□digo del Microchip: CHIP-C5D6E7F7G9H0
Error: El c□digo de microchip ya existe: CHIP-C5D6E7F7G9H0
Por favor, ingrese un c□digo diferente.
C□digo del Microchip:
```

3. Listado de mascotas

```
------------------------------
0) Volver al men□ principal
------------------------------
Ingrese una opci□n: 2

--- Listado de Mascotas ---
Mascota{id=1, eliminado=false, nombre='Fido', especie='Perro', raza='Labrador Retriever', duenio='Ana L□pez', microchip=CHIP-A1B2C3D4E5F6}
Mascota{id=2, eliminado=false, nombre='Michi', especie='Gato', raza='Siam□s', duenio='Juan P□rez', microchip=CHIP-G7H8I9J0K1L2}
Mascota{id=3, eliminado=false, nombre='Coco', especie='Perro', raza='Bulldog Franc□s', duenio='Mar□a Garc□a', microchip=CHIP-M3N4O5P6Q7R8}
Mascota{id=4, eliminado=false, nombre='Luna', especie='Gato', raza='Maine Coon', duenio='Carlos Rodr□guez', microchip=CHIP-S9T0U1V2W3X4}
Mascota{id=5, eliminado=false, nombre='Max', especie='Perro', raza='Pastor Alem□n', duenio='Sof□a Mart□nez', microchip=CHIP-Y5Z6A7B8C9D0}
Mascota{id=6, eliminado=false, nombre='Pipo', especie='Ave', raza='Canario', duenio='Javier S□nchez', microchip=CHIP-E1F2G3H4I5J6}
Mascota{id=7, eliminado=false, nombre='Rocky', especie='Perro', raza='Golden Retriever', duenio='Valentina D□az', microchip=CHIP-K7L8M9N0O1P2}
Mascota{id=8, eliminado=false, nombre='Kira', especie='Gato', raza='Persa', duenio='Pedro Fern□ndez', microchip=CHIP-Q3R4S5T6U7V8}
Mascota{id=9, eliminado=false, nombre='Toby', especie='Perro', raza='Beagle', duenio='Elena Ru□z', microchip=CHIP-W9X0Y1Z2A3B4}
Mascota{id=10, eliminado=false, nombre='Nemo', especie='Pez', raza='Pez Payaso', duenio='Ricardo G□mez', microchip=CHIP-C5D6E7F8G9H0}
Mascota{id=11, eliminado=false, nombre='Cheddar', especie='Perro', raza='Mestizo', duenio='Milagros Barrutia', microchip=CHIP-C5D6E7F7G9H0}
```

4. Eliminación de mascota

```
--------------------------------
Ingrese una opci□n: 5
Ingrese ID de la mascota a eliminar: 1
Mascota y microchip asociado (baja l□gica) eliminados correctamente.
```

Se lista las mascotas para corroborar que la mascota id 1 ya no se encuentra

```
--- Listado de Mascotas ---
Mascota{id=2, eliminado=false, nombre='Michi', especie='Gato', raza='Siam□s', duenio='Juan P□rez', microchip=CHIP-G7H8I9J0K1L2}
Mascota{id=3, eliminado=false, nombre='Coco', especie='Perro', raza='Bulldog Franc□s', duenio='Mar□a Garc□a', microchip=CHIP-M3N4O5P6Q7R8}
Mascota{id=4, eliminado=false, nombre='Luna', especie='Gato', raza='Maine Coon', duenio='Carlos Rodr□guez', microchip=CHIP-S9T0U1V2W3X4}
Mascota{id=5, eliminado=false, nombre='Max', especie='Perro', raza='Pastor Alem□n', duenio='Sof□a Mart□nez', microchip=CHIP-Y5Z6A7B8C9D0}
Mascota{id=6, eliminado=false, nombre='Pipo', especie='Ave', raza='Canario', duenio='Javier S□nchez', microchip=CHIP-E1F2G3H4I5J6}
Mascota{id=7, eliminado=false, nombre='Rocky', especie='Perro', raza='Golden Retriever', duenio='Valentina D□az', microchip=CHIP-K7L8M9N0O1P2}
Mascota{id=8, eliminado=false, nombre='Kira', especie='Gato', raza='Persa', duenio='Pedro Fern□ndez', microchip=CHIP-Q3R4S5T6U7V8}
Mascota{id=9, eliminado=false, nombre='Toby', especie='Perro', raza='Beagle', duenio='Elena Ru□z', microchip=CHIP-W9X0Y1Z2A3B4}
Mascota{id=10, eliminado=false, nombre='Nemo', especie='Pez', raza='Pez Payaso', duenio='Ricardo G□mez', microchip=CHIP-C5D6E7F8G9H0}
Mascota{id=11, eliminado=false, nombre='Cheddar', especie='Perro', raza='Mestizo', duenio='Milagros Barrutia', microchip=CHIP-C5D6E7F7G9H0}
```

5. Se actualiza el dueño de una mascota

```
Ingrese una opci□n: 4
Ingrese ID de la mascota a modificar: 3
Mascota actual:
Mascota{id=3, eliminado=false, nombre='Coco', especie='Perro', raza='Bulldog Franc□s', duenio='Mar□a Garc□a', microchip=CHIP-M3N4O5P6Q7R8}

Deje el campo vac□o para mantener el valor actual.
Nombre (Coco):
Especie (Perro):
Raza (Bulldog Franc□s):
□Modificar fecha de nacimiento? (s/n): n
Due□o (Mar□a Garc□a): Ana Luisa Bordalejo
Mascota actualizada correctamente.
```

Prueba de Roll Back

Se intenta insertar un CHIP duplicado Se genera un Roll Back y la transaccion se detiene

```
Paso 2: Datos del Microchip (Obligatorio)
C□digo del Microchip: CHIP-A1B2C3D4E5F6
Veterinaria (opcional): Patitas
Fecha implantaci□n (yyyy-MM-dd, opcional): 2022-02-16
Error en Service: Duplicate entry 'CHIP-A1B2C3D4E5F6' for key 'uk_microchip_codigo'
Error al crear la mascota: Duplicate entry 'CHIP-A1B2C3D4E5F6' for key 'uk_microchip_cod

Presione ENTER para continuar...
```

CONCLUSIONES

Para finalizar este informe podemos concluir que la organización del proyecto en capas garantiza una organización y una eficiencia del proyecto ya que si algo falla solo se corrige un archivo o una capa y no toda una unidad. El diseño DAO/Service/Main, aunque complejo de implementar inicialmente, demostró ser extremadamente robusto para manejar la lógica de negocio y las transacciones de forma segura y ordenada.

El trabajo en equipo es fundamental para estos proyectos ya que permiten el intercambio de ideas y formas de solucionar los inconvenientes con los que nos enfrentamos. Se puede mejorar la utilización de herramientas como GitHub para mejorar la interacción entre los trabajos individuales.

BIBLIOGRAFIA

- Material de la catedra : Relaciones 1 a 1 en Java.
- https://gemini.google.com/share/78ac0e128923
- https://www.ebiseducation.com/crud-que-es-para-que-sirve-como-funciona-y-ejemplos
- https://lineadecodigo.com/java/conectar-mysql-java/
- https://es.stackoverflow.com/questions/13562/c%C3%B3mo-utilizar-git-para-trabajar-en-un-mismo-proyecto-en-un-equipo-distribuido

HERRAMIENTAS

- **Lenguaje:** Java (JDK 24)
- **Base de Datos:** MySQL 8.0
- **Driver:** MySQL Connector/J 9.5.0
- **IDE:** Apache NetBeans
- **Control de Versiones:** Git y GitHub

Asistencia de IA: Se utilizó un asistente de IA (Gemini) para consultas conceptuales sobre la arquitectura DAO/Service, refinamiento de código y corrección de errores de sintaxis y lógica